CSC 311 – Winter 2022-2023
Design and Analysis of Algorithms

4. Analysis of time efficiency of recursive algorithms

Prof. Mohamed Menai
Department of Computer Science
King Saud University

Outline

- Plan for analysis of recursive algorithms
- Solving recurrences
- Important recurrence types

Plan for Analysis of Recursive Algorithms

- Decide on a parameter indicating an input's size.
- Identify the algorithm's basic operation.
- Check whether the number of times the basic op. is executed may vary on different inputs of the same size. (If it may, the worst, average, and best cases must be investigated separately.)
- Set up a recurrence relation with an appropriate initial condition expressing the number of times the basic op. is executed.
- Solve the recurrence (or, at the very least, establish its solution's order of growth) by backward substitutions or another method.

3

Example 1: Recursive evaluation of $n!$

Definition: $n! = 1.2.....(n\text{-}1).n$ for $n \geq 1$ and $0! = 1$

Recursive definition of $n!$: $F(n) = F(n\text{-}1) * n$ for $n \geq 1$
and
$F(0) = 1$

ALGORITHM $F(n)$

```
//Computes n! recursively
//Input: A nonnegative integer n
//Output: The value of n!
if n = 0 return 1
else return F(n − 1) * n
```

Example 1: Recursive evaluation of $n!$

Size? n

Basic operation? multiplication

Recurrence relation? $T(n) = T(n\text{-}1) + 1$

$T(0) = 0$

Solving the recurrence for T(n)

$$T(n) = T(n\text{-}1) + 1$$
$$T(0) = 0$$

$T(n) = T(n\text{-}1) + 1$

$= (T(n\text{-}2) + 1) + 1 \quad = \quad T(n\text{-}2) + 2$

$= (T(n\text{-}3) + 1) + 2 \quad = \quad T(n\text{-}3) + 3$

…

$= T(n\text{-}i) + i$

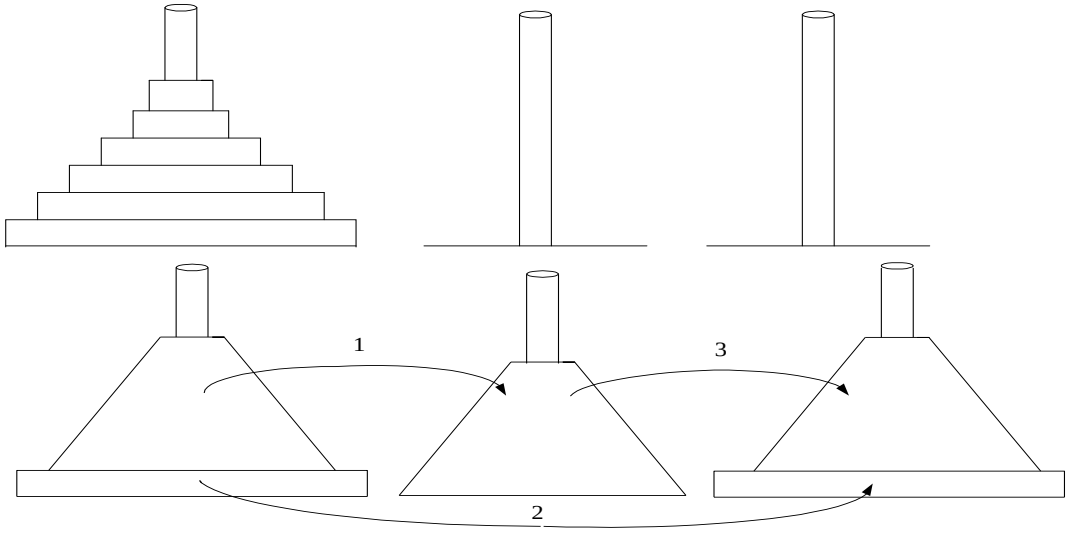
$= T(0) + n$

$= n$

The method is called backward substitution.

6

Example 2: The Tower of Hanoi Puzzle



Recurrence for number of moves: $T(n) = 2T(n\text{-}1) + 1$

$T(1) = 1$

7

Solving recurrence for number of moves

$$T(n) = 2T(n\text{-}1) + 1 \qquad T(1) = 1$$

$$T(n) = 2T(n\text{-}1) + 1$$

$$= 2(2T(n\text{-}2) + 1) + 1 = 2^2T(n\text{-}2) + 2^1 + 2^0$$

$$= 2^2\,(2T(n\text{-}3) + 1) + 2^1 + 2^0$$

$$= 2^3\,T(n\text{-}3) + 2^2 + 2^1 + 2^0$$

$$= \ldots$$

$$= 2^{(n\text{-}1)}T(1) + 2^{(n\text{-}2)} + \ldots + 2^1 + 2^0$$

$$= 2^{(n\text{-}1)} + 2^{(n\text{-}2)} + \ldots + 2^1 + 2^0$$

$$= 2^n - 1$$

8

Example 3: Counting #bits

ALGORITHM *BinRec*(n)

//Input: A positive decimal integer n

//Output: The number of binary digits in n's binary representation

if $n = 1$ **return** 1

else return $BinRec(\lfloor n/2 \rfloor) + 1$

$$\begin{cases} T(n) = T(\lfloor n/2 \rfloor) + 1 \\ T(1) = 0 \end{cases}$$

$$\begin{aligned} T(2^k) &= T(2^{k-1}) + 1 \\ &= \left(T(2^{k-2}) + 1\right) + 1 = T(2^{k-2}) + 2 \\ &\cdots \\ &= T(2^{k-i}) + i \\ &= T(2^{k-k}) + k = k = \log n \end{aligned}$$

9

Fibonacci numbers

The Fibonacci numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, …

The Fibonacci recurrence:

$F(n) = F(n\text{-}1) + F(n\text{-}2)$

$F(0) = 0$

$F(1) = 1$

General 2nd order linear homogeneous recurrence with constant coefficients:

$$aX(n) + bX(n\text{-}1) + cX(n\text{-}2) = 0$$

10

Solving $aX(n) + bX(n\text{-}1) + cX(n\text{-}2) = 0$

- Set up the characteristic equation (quadratic)

$$ar^2 + br + c = 0$$

- Solve to obtain roots r_1 and r_2
- General solution to the recurrence

  if r_1 and r_2 are two distinct real roots: $X(n) = \alpha r_1^n + \beta r_2^n$

  if $r_1 = r_2 = r$ are two equal real roots: $X(n) = \alpha r^n + \beta n r^n$

- Particular solution can be found by using initial conditions

11

Application to the Fibonacci numbers

F(n) = F(n-1) + F(n-2) or F(n) - F(n-1) - F(n-2) = 0

Characteristic equation: $r^2 - r - 1 = 0$

Roots of the characteristic equation: $r_{1,2} = (1 \pm \sqrt{5}) / 2$

General solution to the recurrence: $\alpha \cdot r_1^n + \beta \cdot r_2^n$

Particular solution for F(0) =0, F(1)=1: $\alpha + \beta = 0$

$$\alpha \cdot r_1 + \beta \cdot r_2 = 1$$

12

Important Recurrence Types

- Decrease-by-one recurrences
- A decrease-by-one algorithm solves a problem by exploiting a relationship between a given instance of size n and a smaller size n-1.

  Example: $n!$
- The recurrence equation for investigating the time efficiency of such algorithms typically has the form: $T(n) = T(n\text{-}1) + f(n)$

13

Decrease-by-one Recurrences

- One (constant) operation reduces problem size by one:

  $T(n) = T(n\text{-}1) + c$; $T(1) = d$

  Solution: $T(n) = c(n\text{-}1) + d$ (linear time)
- A pass through input reduces problem size by one.

  $T(n) = T(n\text{-}1) + cn$; $T(1) = d$

  Solution: $T(n) = c[n(n+1)/2\text{-}1] + d$ (quadratic time)

14

Important Recurrence Types

- Decrease-by-a-constant-factor recurrences
- A decrease-by-a-constant-factor algorithm solves a problem by dividing its given instance of size n into several smaller instances of size n/b, solving each of them recursively, and then, if necessary, combining the solutions to the smaller instances into a solution to the given instance.

  Example: binary search
- The recurrence equation for investigating the time efficiency of such algorithms typically has the form

  $T(n) = aT(n/b) + f(n)$

15

Decrease-by-a-constant-factor recurrences

- The Master Theorem

Reading

Chapter 2 (Sections 2.4, 2.5)

Anany Levitin, Introduction to the design and analysis of algorithms, 3rd Edition, Pearson, 2011.